

Self-Join Size Estimation in Large-scale Distributed Data Systems

Theoni Pitoura^{#1}, Peter Triantafillou^{#2}

[#]Research Academic Computer Technology Institute, and University of Patras
26500 Rio, Patras, Greece

¹pitoura@ceid.upatras.gr

²peter@ceid.upatras.gr

Abstract— In this work we tackle the open problem of self-join size (SJS) estimation in a large-scale Distributed Data System, where tuples of a relation are distributed over data nodes which comprise an overlay network. Our contributions include adaptations of five well-known SJS estimation centralized techniques (coined sequential, cross-sampling, adaptive, bifocal, and sample-count) to the network environment and a novel technique which is based on the use of the Gini coefficient. We develop analyses showing how Gini estimations can lead to estimations of the underlying Zipfian or power-law value distributions. We further contribute distributed sampling algorithms that can estimate accurately and efficiently the Gini coefficient. Finally, we provide detailed experimental evidence testifying for the claimed increased accuracy, precision, and efficiency of the proposed SJS estimation method, compared to the other methods. The proposed approach is the only one to ensure high efficiency, precision, and accuracy regardless of the skew of the underlying data.

I. INTRODUCTION

The self-join size of a relation has long been recognized in the DB literature as a key indicator of frequency distributions and, in particular, as an indicator of the skew of the underlying data set. The self-join size of a relation R on an attribute of domain D is $\sum_{i \in D} \alpha_i^2$, where α_i is the frequency of attribute value i in R ([2]). As such, it has played a key role in query optimization strategies. Indeed, some researchers mention that self-join size can uniquely determine the parameter of many distributions ([2]). In addition, self-join size estimation has been used to select appropriate algorithms based on data value skew [14], to estimate errors in estimated intermediate result sizes [22], etc. For these reasons, a large body of literature has focused on developing techniques for estimating the (self-) join size of relations for centralized DBMSs.

However, most if not all of the related work has focused on centralized data systems (or at best on small-scale distributed systems), mainly because, data management research as a whole had not paid particular attention on developing methods and systems that can work efficiently at large/internet scales. Recently, however, there has been a growing number of research on DB-like functionality over large-scale networks and peer-to-peer (p2p) systems ([1], [21], [26], etc), focusing on complex query processing and

optimization. Query optimization would gain from distributed (self-)join size estimation techniques, just like in the centralized case, but these techniques have not been studied yet. Moreover, distributed self-join size estimation would be highly beneficial in many other applications of large-scale data networks. One example, *web clustering* ([18]), comes from distributed data mining, where in particular self-join sizes can help determine the similarities between two or more (distributed) collections of web pages, needed when deciding whether to cluster these collections. Another application area is distributed information retrieval (IR) and in particular p2p web searching. A significant body of recent works is dealing with the issue of query routing: deciding which (possibly distributed) document collections are most appropriate for a particular keyword query. Specifically, given a collection (partial result), the goal is to identify new collections that improve the result quality for the query. This involves identifying the size of the overlap between candidate collections and the partial result ([4]). Viewing these collections as (distributed) relations with one attribute being *docID*, this is equivalent to estimating the join size on *docID* of the candidate relations with the relation representing the partial result. It is well-known ([2]) that knowing the self-join sizes of two relations can be used to provide an upper bound for their join size; hence, obviously self-join size estimation of distributed relations can prove highly beneficial.

Most of the work in estimating the (self-) join size of relations for centralized DBMSs can be classified into parametric, sampling, or statistical metadata-dependent algorithms. Parametric algorithms ([8], [25], etc) assume knowledge of the underlying distribution of the data, which they exploit for their estimation. Sampling-based techniques ([23], [24], [17], [12], etc) repeatedly probe the database, obtaining samples which they use to estimate the self-join size, after scaling appropriately. Statistical metadata-dependent approaches ([2], [3], [22], [10], etc) develop and maintain statistical metadata (such as histograms, sketches, etc) for the involved data sets, which they exploit for their estimation. Finally, there are hybrid algorithms ([13], [3]) where sampling is used to build summary statistics.

A p2p data system is a typical representative of large-scale distributed data systems. Hence, hereafter we will be using these terms interchangeably. Such data systems offer

some unique challenges when attempting to perform self-join size estimations: their high dynamics and the goals of high efficiency, low overhead, high estimation accuracy, and scalability place difficult optimization and trade-off decisions. Thus, we believe that building and maintaining statistical metadata structures is likely to incur a high cost. Parametric algorithms, on the other hand, could be used, since it is widely accepted that for most web and/or peer-to-peer applications, data value distributions follow either a power-law distribution, or its close relative, Zipf's law ([29], [5]). However, given the environment's high dynamics, the algorithms must be robust in terms of offering high accuracy under varying conditions, such as data skews. Our approach assumes an underlying Zipfian or power-law distribution and utilizes sampling techniques (i.e. it is a hybrid approach). Furthermore, even with a sampling technique, sampling sizes must be small, or the communication costs will become dominant and estimation strategies for query optimization purposes, for example, will not be very useful since query costs will be dominated by (self-) join size estimation costs. So, the open problem is whether we can develop self-join size estimation techniques in a p2p environment that are highly accurate, do not hurt scalability, impose little execution overhead, and are robust (i.e. perform well under changing data characteristics, such as skews). This is the problem tackled in this paper.

The paper is organized as follows: Section 2 presents background material and definitions, and describes related work, the problem formulation and our contributions. The fundamental theorems upon which our novel G-based estimation algorithm is based are defined and proved in section 3. In section 4 we describe our proposed algorithm, and develop adaptations of five well-known (self-) join size estimation algorithms to a p2p environment. In section 5 we present results from comprehensive experimentation, and, in section 6, we conclude and discuss future work.

II. BACKGROUND, PROBLEM FORMULATION AND CONTRIBUTION

A. Lorenz Curves and the Gini coefficient

We use *Lorenz curves* - originally used in economics and ecology ([9]) - to map the cumulative percentage of tuples (x-axis), ordered by their attribute value, with the cumulative percentage of their attribute value (y-axis). The greater the distance is from the diagonal line, the greater the inequality of the value distribution becomes. If all tuples have the same attribute value, the curve is a straight diagonal line, the *line of perfect equality* (i.e. $p\%$ of the tuples have $p\%$ of the total sum of values, for $0 \leq p \leq 1$). If there is any imbalance, the curve falls below this line.

The total amount of value inequalities can be summarized by the Gini coefficient (G) ([9]). G expresses the ratio of the area enclosed by the line of perfect equality

and the Lorenz curve, over the total triangular area under the line of perfect equality.

Definition 1 - The Gini coefficient (G) is defined as the ratio of two geometrical areas: a. the area between the line of perfect equality and the Lorenz curve (area A in Fig. 1), and b. the area under the 45 degree line (areas $A+B$):

$$G = \frac{A}{A+B} \quad (1)$$

Because areas $A+B$ represent half of the unit box, we have $A+B = \frac{1}{2}$, $G = \frac{A}{A+B} = 1 - 2 \cdot B$. Therefore, to calculate G , we only need to calculate area B .

Let us assume that we have a sequence of m values (here, a tuple's join attribute value), say y_1, y_2, \dots, y_m , generated by a probability distribution on a domain $D = [u_{min}, u_{max}]$, such that: $y_1 \leq y_2 \leq \dots \leq y_m$.

If the probability distribution is *discrete*, the area below the Lorenz curve is equal to the area of the polygon joining the points $(i/m, L_i/L_m)$, where $i=0,1,\dots,m$, $L_0=0$,

$L_m = T = \sum_{j=1}^m y_j$ (Fig. 1); T expresses the sum total. Thus,

if (F_i, L_i) is the point in the Lorenz curve which corresponds to value y_i , then $F_i = i/m$ (the cumulative proportion of all values), and $L_i = \frac{1}{T} \sum_{j=1}^i y_j$ (the proportion

of the sum total corresponding to $F_i\%$ of all m values).

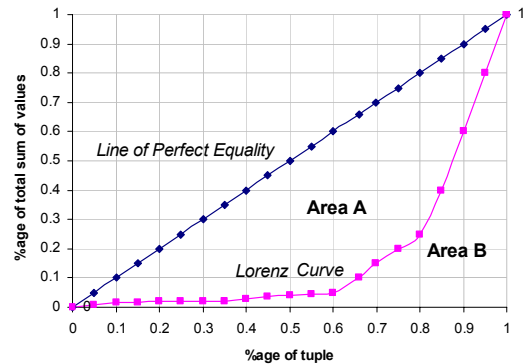


Fig. 1 Lorenz Curve and the Gini Coefficient in a discrete value distribution

Alternatively, the Gini coefficient ([9]) can be defined as:

Definition 2 - The Gini coefficient (G) is defined as the relative mean difference, i.e. the mean of the difference between every possible pair of values divided by their mean, such as:

$$G = \frac{1}{2\mu_y} \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m |y_i - y_j|, \quad \mu_y = \frac{1}{m} T = \frac{1}{m} \sum_{j=1}^m y_j \quad (3)$$

B. Related work

Related work concerns existing techniques used to estimate (self-)join sizes in data managements systems. As

we have already discussed, these techniques can be categorized into *parametric*, *sampling-based*, *statistical metadata-dependent* and *hybrid*. Most of them are used to build data statistics summaries, whereas sampling can also be used to estimate a database metric, such as the query result size, by collecting a small and usually, random sample from the base data and scaling the result.

In general, data summarization differs from sampling in that the former is mainly interested in the representation of data distributions whereas the latter in the collection of them. However, in some research work, either sampling is used to collect information about the data distribution in order to build data synopsis ([2], [11], [13]), or statistics summaries on frequency distribution are used to guide sampling to estimate join query results [7]). In our paper, we use sampling to estimate the skew of the data distribution and then, use this information to estimate the self-join size. Thus, data summarization techniques have different goals than our estimation sampling algorithm. Also, using sampling, we do not need to store and maintain summary statistics, which may add great maintenance costs.

There are various sampling techniques used to estimate (self-)join, or, (in general) query size in centralized data management systems: *sequential* ([17]), *adaptive* ([24], [23]), *bifocal* ([12]), *clustered* ([20]), *independent* ([15], [16]), *cross* and *cross-product* ([15], [16]), and *double samplings* ([19]), and the *sketch-based randomized approximation* algorithms ([2], [3]). In addition, [14] uses several sample-based estimators to estimate the number of distinct values of an attribute in a relation, but this work is complementary to the query size estimation problem.

Clustered sampling ([20]) involves randomly selecting a set of disk blocks of one relation and a set of disk blocks of the other, then comparing all tuples within these blocks to check if they join, and finally estimating the solution based on the number of samples, the size of the relations and the sample sum. Haas et al ([15]) categorize sampling algorithms for join size estimation into groups based on: (a) whether the sample unit is a tuple, or a memory page composed of tuples, and (b) whether sampled items are compared with all tuples in the join relation (*independent sampling*), or with all tuples occurring in samples taking from these join relations (*cross-sampling*), or with only those tuples taken in the most recent sample of the join relations (*cross-product sampling*). *Double sampling* ([19]) is an approach for a fixed-precision estimation which uses a pilot sample to compute an estimate for the mean and the variance of the sample sizes. They use this estimate to refine the termination criteria of the basic (adaptive) sampling. Among the existing techniques, we chose *sequential* ([17]), *adaptive* ([24], [23]) *bifocal* ([12]), and *cross-samplings* ([15], [16]), and a sketch-based approach presented in [3] (and as *sample-count* in [2]) to adapt into a p2p environment as the best competitors to our proposed algorithm.

To our knowledge, this is the first work that deals with estimating self-join size estimation in p2p data systems.

C. Problem formulation

We assume a single-attribute relation $R(A)$, or simply R , with attribute A (for a multi-attribute relation we would choose one of the attributes, A as a (self-) join attribute). R has a very large number of tuples, say $M \gg 0$, that are uniformly distributed over the nodes of a p2p network. We assume an N -node p2p network, and that each node p_i , with $i \in \{1, 2, \dots, N\}$ is immediately connected with a number of nodes, its *neighbors*. We make no further assumptions for the type of the p2p network. Further, in general, we may not have any knowledge as to which nodes store which values.

We assume that attribute A of relation R is used as a single-attribute index of a tuple $t \in R$, and that A is characterized by its value domain, $D = [u_{\min}, u_{\max}]$. Each tuple is uniquely identified by a primary key, $t.key$.

Definition 3 – For a relation R , the self-join size on its attribute A of domain D is:

$$SJS(A) = \sum_{i \in D} \alpha_i^2 \quad (4)$$

where α_i is the frequency (i.e. the number of appearances) of attribute value i in R ([2]).

D. Our contributions

Our contributions are:

- We develop adaptations of five well-known (self-) join size estimation techniques (coined *sequential*, *cross-*, *adaptive*, *bifocal* samplings, and the sketch-based approach, *sample-count*) that can be deployed over a p2p environment. The central question here is whether by simply borrowing and adapting existing algorithms suffices to solve the problem at hand. The answer to this is negative.
- We develop a novel technique to estimate self-join sizes based on the Gini coefficient (G) in a p2p network. We show that G can be estimated with a relatively small sample size and with a very small estimation error.
- We contribute analyses of how G is related with the skew parameter of Zipfian distributions, and how self-join sizes can be estimated using these skew parameter values.
- We implement all above algorithms and conduct detailed experimentation to study their performance in a simulated p2p data system, with Zipfian attribute values distributions of various skews. Our results testify for the superiority of the G-based approach and substantiate our claims of low overheads, high estimation precision and high accuracy, regardless of skew.

III. GINI-BASED ESTIMATION OF SJS

In this section we present the fundamental theorems upon which our novel estimated algorithm is based. Specifically, we show how G estimations can lead to estimations of the parameter of Zipfian distribution, and, in turn, to estimations of the self-join size of a relation whose value distribution follows Zipf's law.

A. Zipf's law and the Gini coefficient

Zipf's law may be stated mathematically by the discrete probability distribution function $f(u'_i) = Cl^{-\theta}$, which holds for any value $u'_i \in D$ of d values, where θ is the skew parameter ($\theta > 0$), l the value's rank, and C a constant equal to $C = \left(\sum_{j=1}^d j^{-\theta} \right)^{-1}$. In the following paragraphs we prove a

formula which relates G of any sequence of Zipfian distributed values with θ and, thus, enables us to calculate G given θ , and vice-versa.

Let us first assume that we have a sequence of m values, say y_1, y_2, \dots, y_m , generated by a Zipfian distribution on an integer-value domain $D = [1, d]$, such that: $y_1 \leq y_2 \leq \dots \leq y_m$. The corresponding Lorenz curve is plotted in Fig. 2.

Theorem 1. *The Gini coefficient (G) of any sequence of values produced by a value distribution on a domain D which follows Zipf's law depends only on the skew parameter of the distribution, θ and, D . Specifically, G , θ and $D = [1, d]$ are related by the following equation:*

$$G = 1 - \frac{2}{\sum_{j=1}^d j^{-\theta} \sum_{j=1}^d j^{-\theta+1}} \sum_{k=1}^d \left[k^{-\theta} \sum_{j=1}^{k-1} j^{-\theta+1} + \frac{1}{2} k^{-2\theta+1} \right] \quad (5)$$

Proof of Theorem 1. We assume that a_k is the number of occurrences of value $u_k \in D$ in our sequence of m values, for $k=1, 2, \dots, d$, where $a_k \geq 0$. This means that value u_1 appears a_1 times, value u_2 appears a_2 times, and so on.

Then, $\sum_{k=1}^d a_k = m$ (6). Furthermore, if $f(u_k)$ is the probability distribution function, then: $\alpha_k = mf(u_k)$ (7).

Without loss of generality, we assume that $u_1 \leq u_2 \leq \dots \leq u_d$, and $u_k = k$. Because our sequence of m y -values generated from domain D is increasingly ordered, a_k occurrences of value u_k lie together in consecutive y 's, starting, from y_{i+1} to y_{i+a_k} , assuming that there are i distinct y -values lower than u_k .

Then, the x -axis of the Lorenz curve can be split into a maximum of d consecutive subranges, each equal to a_k/m , for $k=1, 2, \dots, d$. The corresponding k^{th} subrange in the x -axis extends from $F_i = \frac{y_i}{m}$ to $F_{i+\alpha_k} = \frac{y_{i+\alpha_k}}{m}$, and in the y -axis

from $L_i = \frac{1}{T} \sum_{j=1}^{k-1} \alpha_j u_j$ to $L_{i+\alpha_k} = \frac{1}{T} \sum_{j=1}^k \alpha_j u_j$, where i is the number of previous subranges. (Note that, for $k=1, i=0$ and $F_0 = L_0 = 0$). Then, area B is composed of a number of trapeziums, one for each one of the maximum d consecutive subranges of equal y -values. We can calculate the area of each trapezium B_i , as marked in Fig. 2, as:

$$B_k = \frac{1}{2} (L_i + L_{i+\alpha_k}) \cdot \frac{\alpha_k}{m} = \frac{1}{2mT} \left[2 \sum_{j=1}^{k-1} \alpha_j u_j + \alpha_k u_k \right] \alpha_k \quad (8)$$

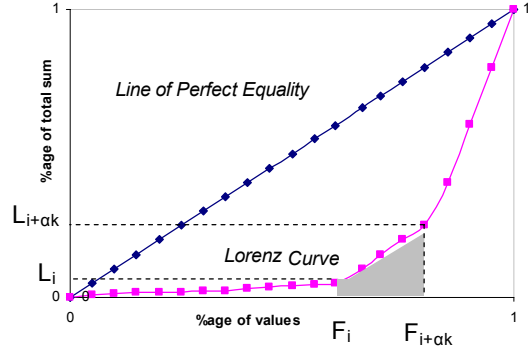


Fig. 2 The Lorenz Curve of a Discrete Zipfian Distribution

Since our y -values follow a Zipfian distribution on domain $D=[1,d]$, for any $u'_i \in D$, $f(u'_i) = Cl^{-\theta}$ (9), where θ is the skew parameter ($\theta > 0$), l is the rank of its frequency, and C is equal to $C = \left(\sum_{j=1}^d j^{-\theta} \right)^{-1}$ (10) (a normalization constant). It is obvious that: $f(u'_1) \geq f(u'_2) \geq \dots \geq f(u'_d)$.

Assumption 1: We assume that $u'_i = u_l = l$, i.e. value 1 is the most frequent in our sequence, value 2 the second most frequent, value u_l is the l^{th} most frequent, and so on. This assumption defines a mapping between values of domain D and their order of frequency. This assumption does not lose generality in cases when the values themselves are insignificant and we only care about their frequencies of appearance. This is the case of the self-join size estimation that we examine here.

Thus, equation (8) using (7) and (9) becomes:

$$B_k = \frac{1}{mT} \left[\sum_{j=1}^{k-1} mf(u_j) u_j + \frac{1}{2} mf(u_k) u_k \right] mf(u_k) \Rightarrow$$

$$B_k = \frac{Cm}{T} \left[\sum_{j=1}^{k-1} Cj^{-\theta} j + \frac{1}{2} Ck^{-\theta} k \right] k^{-\theta} \Rightarrow$$

$$B_k = \frac{C^2 m}{T} \left[k^{-\theta} \sum_{j=1}^{k-1} j^{-\theta+1} + \frac{1}{2} k^{-2\theta+1} \right] \quad (11)$$

The total sum of values, T using equation (2) and the definition of a_k becomes:

$$T = \sum_{j=1}^m y_j = \sum_{j=1}^d \alpha_j u_j = \sum_{j=1}^d mf(u_j) u_j = \sum_{j=1}^d mCj^{-\theta} j = mC \sum_{j=1}^d j^{-\theta+1} \quad (12)$$

Since area B is composed by d trapeziums:

$$B = \sum_{k=1}^d B_k \quad (13)$$

Then, by definition 1, G can be expressed geometrically using equations (13), (11), and (12), as:

$$G = 1 - 2 \cdot B = 1 - 2 \sum_{k=1}^d B_k = 1 - \frac{2C}{\sum_{j=1}^d j^{-\theta+1}} \sum_{k=1}^d [k^{-\theta} \sum_{j=1}^{k-1} j^{-\theta+1} + \frac{1}{2} k^{-2\theta+1}]$$

Replacing C by (10), we have equation (5). \square

We have proved a similar theorem for Power Law distributions, but we omit it due to space limitations.

B. Self-Join Size and the Gini coefficient

Now, using *Assumption 1* (used in the proof of theorem 1, i.e. that having a Zipfian distribution, a value u_i of domain $D=[1, d]$ is equal to integer i and to its frequency rank ($u_i = i$)), we prove the following theorem:

Theorem 2. *The self-join-size of a relation R on an attribute A of domain $D=[1, d]$ whose values, u_i follow a Zipfian distribution with parameter θ , are related by the following equation:*

$$SJS(A) = m^2 \sum_{i=1}^d i^{-2\theta} / \left(\sum_{i=1}^d i^{-\theta} \right)^2 \quad (14)$$

Proof of Theorem 2. We have already seen above that, if $f(u_k)$ is the probability distribution function, then: $\alpha_k = mf(u_k)$. Thus, from definition 3,

$SJS(A) = \sum_{i \in D} \alpha_i^2 = \sum_{i=1}^d (mf(u_i))^2 = m^2 \sum_{i=1}^d f^2(u_i)$, and since $f(u_k)$ is a Zipfian function, from equation (9) we have:

$SJS(A) = m^2 C^2 \sum_{i=1}^d i^{-2\theta}$. Replacing C in equation (10), the

theorem is proved. \square

Hence, estimating G leads to the Zipfian parameter from which (Theorem 2) we obtain a SJS estimation.

IV. P2P SJS-ESTIMATION ALGORITHMS

In this section we present a novel technique to estimate SJS in a p2p network based on G , using the theorems presented before. In addition, we adapt five well-known sampling-based algorithms that deal with the problem of estimating (self-)join size in a centralized relational database, namely (a) sequential ([17], [18]), (b) cross ([15]), (c) adaptive ([23], [24]), and (d) bifocal samplings ([12]), and (e) count-sample using random sketches ([3], [2]). To our knowledge, there is no previous work attempting to

develop, or adapt these algorithms in a distributed or p2p environment.

The unit of sampling is a node (often referred as *node-sample* in this paper) and we collect the tuples that a node contains. We do not examine the tuple as a sampling unit since it would be extremely expensive in terms of the required number of messages (a.k.a. hops).

There are various sampling techniques we could apply to select the values needed to estimate SJS. Here we present random walking, which can be widely applied to any P2P network, structured or not (also used in [28]): a node randomly selects one of its neighbors (using the function `randomNeighbor()`) to forward the sampling message, until the required number of samples has been reached.

A. Algorithm G -based sampling (GBS)

The main steps of our algorithm are: first estimate G using a sample of the attribute values (using definition 2 on the sampled values), then, given G , estimate the skew parameter θ of the Zipfian distribution (using theorem 1), and finally estimate SJS from θ (using theorem 2).

Our estimation is based on G , a statistics metric which captures both inequalities and skewness of an attribute whose value distribution follows Zipf's law. The key here is to show that G can be accurately estimated with only a small sampling size, since the sampling size directly affects the communication costs for computing G . Empirical studies and extended experimentation has shown that G is a metric that can be estimated with high accuracy demanding only a small sample size ([28]).

Algorithm *GBS* works as follows: A node p_{init} initiates the algorithm, and sends a *GBS* message to one of its neighbors, say p_i , selected using the function `randomNeighbor()`. Upon receiving the message, node p_i generates a structure T_i and sends it, together with the number of its tuples, m_i , to node p_{init} . Structure T_i stores the distinct join attribute values of p_i 's tuples together with the number of their appearances. Then, if sampling has not finished, p_i forwards a similar *GBS* message to one of its neighbors randomly selected, and the process is repeated. Node p_{init} receives all messages sent from nodes p_i together with T_i and m_i , and merges structures T_i into a structure T . When the count of all m_i reaches the expected number of samples (say m_{max}), node p_{init} estimates SJS using the function $g_based(T)$ which first estimates G based on T , then θ based on G , and finally SJS based on θ . Thus, $SJS_{GBS} = g_based(T)$.

Sampling finishes when the number of sampled tuples, m reaches m_{max} (it corresponds to the sanity bound $m < m_{max}$). We assume that m_{max} is determined by sampling or by known statistics, as in centralized versions.

B. Algorithm P2P sequential sampling (pSEQ)

A generic sequential sampling algorithm ([17], [18]) collects values sequentially, estimates the requested metric (here, SJS), and if the number of samples, or the estimated

metric satisfies a specific condition (it has reached a ‘safe’ value), it terminates.

Algorithm $pSEQ$ is similar with algorithm GBS , except the method they use to estimate SJS. In $pSEQ$, p_{init} uses function $sjs_calcu(T)$ to calculate the (true) SJS of the merged structure T , say S_T . Then, $SJS_{pSEQ} = M/m \cdot S_T$. Recall that M is the total number of tuples of the relation.

In this p2p version of the sequential sampling algorithm, the termination condition depends on the number of tuples sampled. Alternatively, it could depend on the estimated metric, as assumed in [17] (this is the case presented in the p2p adaptive sampling algorithm below). However, our choice does not affect the performance evaluation of the algorithms presented here.

C. Algorithm P2P cross sampling (pCROSS)

Cross sampling ([15], [16]) is usually used for join size estimation of two (or more) relations. The idea is to select a random sample from both relations, compute the join size of the samples, and then scale the result appropriately, i.e. multiply by the factor $(M_1 M_2 / m_1 m_2)^2$, where M_1, M_2 are the total number of tuples of the relations and m_1, m_2 their corresponding sample sizes. In the case of SJS estimation, there is one relation and one sample (i.e. $M_1=M_2$ and $m_1=m_2$). $pCROSS$ is exactly the same with $pSEQ$, except the equation to estimate SJS, which is:

$$SJS_{pCROSS} = (M/m)^2 S_T.$$

D. Algorithm P2P adaptive sampling (pADA)

The main idea in adaptive sampling ([23],[24]) is to partition the query size into x disjoint subsets. The algorithm repeatedly and randomly selects one of these subsets, computes its size, and adds it to a current sum, say S . After k such subsets, the algorithm terminates and estimates the size of the query as $(x/k)S$. The termination condition depends on k and S .

The idea of algorithm $pADA$ is the following: Assuming that the sample unit is a node, we need N^2 total subsets (partitions), since each node participates in N subsets in a network of N nodes. Specifically, let R_p be the set of tuples (i.e. part of R) that corresponds to node p . Then, $SJS = |R \bowtie R| = \bigcup_{p_i, p_j} \{R_{p_i} \bowtie R_{p_j}\}$, for $i, j=1, 2, \dots, N$. Thus,

when n nodes are sampled, the corresponding number of partitions is: $\sum_{j=1}^n (2j-1)$. This is because when the n^{th} node is

sampled, there are $2n-1$ additional partitions selected: one partition $R_{p_n} \bowtie R_{p_n}$, $n-1$ partitions $R_{p_n} \bowtie R_{p_j}$, and $n-1$ partitions $R_{p_j} \bowtie R_{p_n}$, for each one of the previously sampled nodes p_j , for $j=1, 2, \dots, n-1$.

Thus, algorithm $pADA$ works as follows: Nodes repeatedly send a $pADA$ message to one of its neighbors, using $randomNeighbor()$. Each node p_i calculates its SJS,

say S_i , based on its tuples, and, sends S_i to node p_{init} until the termination conditions have been reached, based on the number of nodes currently sampled, say n' , and the SJS of these nodes, say $S_{n'}$. When the termination conditions have been fulfilled, p_{init} calculates SJS by:

$$SJS_{pADA} = N^2/k S_{n'}, \text{ where } S_n = \sum_{i=1}^n S_i, \text{ and } k = \sum_{i=1}^n 2i-1$$

which is the total number of the partitions that correspond to n .

For the termination conditions, $pADA$ uses two variables: s_flag which corresponds to the termination condition $S > b \cdot f(p, \epsilon)$ of the centralized adaptive algorithm, and n_flag which corresponds to the sanity bound $n < n_max$, where n_max is the maximum sample size. It is assumed that parameters b and n_max can themselves be determined by sampling or known statistics (as in the centralized version). Function $f()$ depends on the desired confidence and inaccuracy parameters p and ϵ respectively ([12]). Please, note that, in the centralized version, the first termination condition is based on the SJS of the current partitions selected, whereas, here s_flag is calculated from the SJS of the nodes selected, since, otherwise it would be highly expensive to communicate the necessary information to all nodes sampled.

E. Algorithm P2P bifocal sampling (pBIF)

Bifocal sampling ([12]) was proposed to estimate the size of an equi-join of two relations, being more efficient in skewed data distributions. It categorizes values (and tuples) in each relation into two groups, *dense* and *sparse*, based on the number of tuples with the same join attribute value. According to this grouping, it employs two different sampling procedures: one *dense-dense-estimation* and one *sparse-any-estimation*. The former estimates the size of all dense-dense subjoins, and the latter the sum of the sizes of all sparse-dense and sparse-sparse subjoins. The main idea is that a dense value has many tuples in both relations and, with high probability will be represented by a join performed in a small sample size. On the other hand, when one of the values of a sample is sparse and the other is dense, if sampling is executed from the dense side, the sparse values are not likely to be missed (for that, sparse-any-estimation algorithm is applied twice). However, for sparse-sparse subjoins, individual subjoins may be missed.

Since here we estimate the self-join size of a relation, we only have the cases of dense-dense and sparse-sparse subjoins, and we apply the sparse-any-estimation procedure once. Specifically, $pBIF$ works as follows:

Node p_{init} initiates and sends a $pBIF$ message to one of its neighbors using the function $randomNeighbor()$. Node p_{init} needs two parameters: a threshold δ to characterize dense values, and n_max equal to the maximum sample size, which both are forwarded to the neighbors. Each node p_i which receives a $pBIF$ message, first calculates $mult_T(u)$

for each value u in T , which T (also described before) is a structure with all the distinct join attribute values of all tuples of all nodes currently sampled. $\text{mult}_T(u)$ is the number of appearances of value u in T , as defined in [12]. Depending on $\text{mult}_T(u)$, node p_i calculates the (current) self-subjoin sizes of dense, S_d and sparse values S_s . Specifically,

```

for each value u in T do {
  if  $\text{mult}_T(u) \geq \delta$  // dense value
  then  $S_d = S_d + (\text{mult}_T(u))^2$ ;
  else if  $\text{mult}_T(u) < M/m$  // sparse value
  then  $S_s = S_s + (\text{mult}_T(u))^2$ ;
}
```

When the requested number of nodes has been reached (i.e. n_{max}), the last sampled node sends to node p_{init} the triple (S_d, S_s, m) , which corresponds to the self-subjoin sizes of dense (S_d) and sparse (S_s) values of all m tuples of the nodes sampled. Then, $SJS_{pBIF} = (M/m)^2 S_d + M/m S_s$.

Please, note that in *pBIF* a node sends to its neighbor the merged structure T , whereas in the previous algorithms, a node would send only its own structure T_i to p_{init} . This costs more in bandwidth (heavier messages), but otherwise we would loose in accuracy of SJS estimation.

For efficiency (i.e. fewer messages), we perform dense-dense-estimation and sparse-sparse-estimation (as explained before, the only case is a sparse-sparse self-subjoin sizes) on the same sample of n nodes. Similarly with the centralized bifocal algorithm, while estimating dense-dense self-subjoin sizes, values that appear less than δ times in the sample are ignored, whereas, while estimating sparse-sparse self-subjoin sizes, dense tuples are ignored.

F. Algorithm P2P sample-count (*pSCOUNT*)

Another approach designed to estimate self-join sizes is a sketch-based approach presented in [3] (an application of this approach for tracking self-join sizes in limited storage is also presented in [2], where it is called ‘sample-count’). The algorithm estimates SJS of a sequence of M values with the median, Y of s_2 random variables Y_1, Y_2, \dots, Y_{s_2} . Each Y_i is the average of s_1 random variables $X_{ij} : 1 \leq j \leq s_1$, which are independent and identically distributed. Each X_{ij} is computed from the sequence as follows:

- Choose a random member ap of the sequence, $ap = l$.
- Let r be the number of occurrences of l among the members of the sequence following ap (inclusive).
- Define $X = M(2r-1)$.

In *pSCOUNT* we estimate the SJS of the tuples of the nodes sampled by random walking and then scale to the whole network. We set s_2 to be the number of nodes sampled, and s_1 the (average) number of tuples per node sampled. We also define an order on the (join attribute values of the) selected tuples, needed to compute r in step ii above. We say that “a tuple t_q follows a tuple t_p ”, when:

a. node pp was selected earlier than node pq (in random walking), if tq and tp are stored at different nodes, pq and pp respectively, and

b. tuple tp is selected earlier than tuple tq , if both tuples are stored at the same node (tuples may be selected randomly, according to their time of insertion, etc).

Algorithm *pSCOUNT* is slightly different than the other p2p algorithms. Messages are forwarded to the neighbors till s_2 nodes are selected, and, then, are sent back, following the reverse route, together with the merged structure, T (i.e. at each node p_i , structure T contains information from all tuples of the nodes selected after node p_i including itself). Additionally, each node p_i sends to node p_{init} a value Y_i , calculated as follows:

- for every tuple t_k of node p_i with value u_k {
- $r_k =$ the number of occurrences of u_k in T
- $X_{ik} = m_i * (2 * r_k - 1)$ }
- $Y_i =$ average of all X_{ik}

When node p_{init} receives all s_2 messages with s_2 values Y_i , it estimates SJS with the formula: $SJS_{pSCOUNT} = (M/m)^2 Y$, where Y is the median of s_2 values Y_i . Scaling is performed with the factor $(M/m)^2$, similarly to *pCROSS*.

V. EXPERIMENTAL RESULTS

A. Evaluation criteria

In this section we report on extended experimental performance evaluation of the algorithms presented above. The results are averaged over $u=100$ different samples (i.e., 100 different runs). The evaluation criteria we used are:

a. *accuracy* (expresses the degree of conformity of estimated SJS to its true value), measured by the *percentage (%age) error*, is defined as the average of the absolute difference of the estimate and the true SJS, divided by the estimate;

b. *precision* (expresses the degree to which further SJS estimations will show the same or similar results), measured by the *relative variance*, is defined as the average square of the difference between the estimate and the *expected value of the estimate*, divided by the square of the *expected value of the estimate* (the expected value of the estimate is the average estimate of all different samples/runs of algorithm);

c. *performance* (expresses the efficiency of executing these algorithms under the wide-scale distribution setting of a p2p data management system), measured by the number of hops among the nodes of the network.

In addition, the centralized estimation sampling algorithms usually use termination criteria when a pre-specified sample size or accuracy has been reached. Thus, others terminate with a lower sample size, and thus the accuracy of estimation is low, and others terminate with higher sample sizes, and thus, their performance is low (i.e. a higher message count). For this reason, we do not use termination criteria but we assume that all algorithms are executed with a maximum sample size, and compare the accuracy of their estimation for various sample sizes.

We used the node as the sampling unit (i.e. node-samples). In the results presented below, N expresses the number of nodes, n the sample size, M the number of tuples of the relation and m the number of tuples sampled.

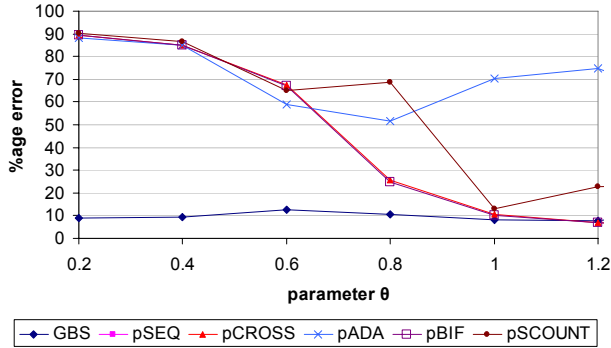


Fig. 3 Accuracy of various SJS estimation sampling algorithms for various θ s for a sample size $n=10$ ($N=1000$)

B. Experimentation setup

We simulated a p2p data management system running on a N -node Chord ([30]) overlay, with $N = 1000, 4000,$ and 10000 nodes. There was one single-attribute relation, with 100000 tuples distributed across all nodes. The attribute values were generated by Zipfian distributions over the integer domain $[1, 10000]$, with parameter θ ranging from 0.2 to 1.2 (this covers results of studies on file and web sites popularities, such as [6], [27], [29]).

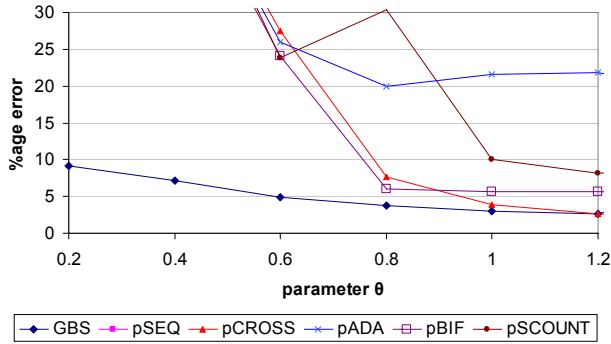


Fig. 4 Accuracy of various SJS estimation sampling algorithms for various θ s for a sample size $n=50$ ($N=1000$)

C. Results

1) *Accuracy*: In Fig. 3 and Fig. 4, the *percentage (%age) error* for each one of the algorithms is illustrated for node-sample sizes $n=10$ and 50 , respectively, for various values of θ and for $N=1000$ nodes (results are similar for other values of N). *pSEQ* is ‘tragically’ inaccurate (it has more than 100% error), and thus, it does not appear in the figures. *GBS* has the highest accuracy (i.e. the lowest %age error) among all algorithms. The accuracy of *pBIF* is improved for higher θ s. This is expected, since the centralized bifocal is more efficient for skewed distributions. *pCROSS* has similar accuracy with *pBIF*

especially for low sample sizes. For low and medium values of θ ($\theta < 1$), *GBS* has significantly higher accuracy than the others, with a relative error around 10% for a small sample size $n=10$, and ranging from 3% ($\theta=1.2$) to 9% ($\theta=0.2$) for $n=50$. *pCROSS*’s and *pBIF*’s accuracy is more than 20%, for $\theta < 1$. For high θ s, these two algorithms have a comparable accuracy with *GBS* (but their accuracy is still lower). Thus, *GBS* is the only algorithm that can guarantee excellent accuracy regardless of skew: for all different θ s, it ensures a very high accuracy.

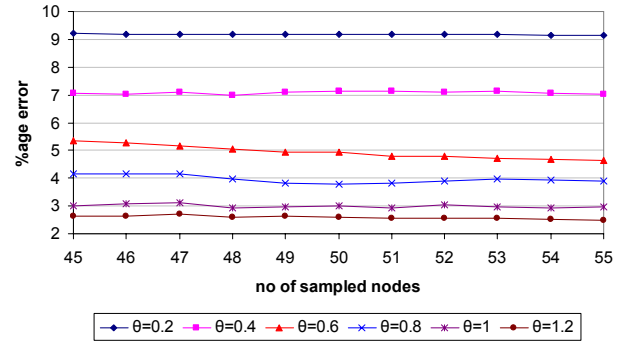


Fig. 5 Accuracy (expressed in %age error) of our G -based estimation algorithm for a number of samples around 5214 (for $n=52, m=5186.39$)

In addition, we compare the accuracy results of *GBS* with the *centralized* versions of cross-sampling, adaptive, and bifocal, as presented in [12]. Bifocal has a %age error around 2 when the join size is $\Omega(\sqrt{M} \lg M)$, which in our experiments is true for $\theta \geq 0.8$ and for a sample size $O(\sqrt{M} \lg M)$, which here corresponds to ≥ 5214 tuples. The %age error metric of *GBS* for an equal number of samples (which corresponds to a node-sample equal to 52) is illustrated in Fig. 5. We notice that the results of our algorithm are comparable to the centralized ones: the %age error ranges between 2.5% and 9% for $\theta=0.2-1.2$, which is significantly lower than that of the other centralized algorithms, except bifocal (their lower %age error is around 10 for $\theta=1$, and >40 for other values of θ).

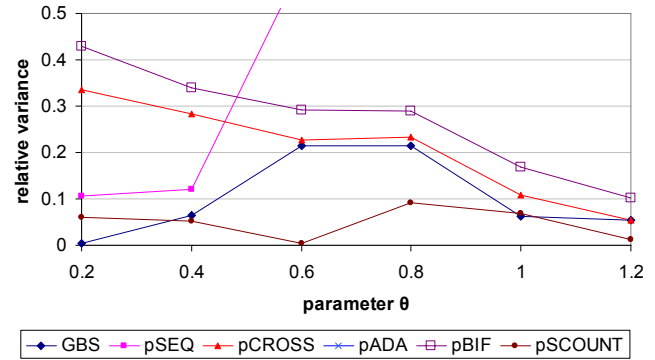


Fig. 6 Precision of various SJS estimation sampling algorithms for various θ s for a sample size $n=50$

TABLE I
PERFORMANCE IN TERMS OF THE NUMBER OF HOPS FOR DIFFERENT ZIPFIAN DISTRIBUTIONS, FOR A %AGE ERROR < 30, 20, 10, AND FOR $N=1000, 4000, 10000$ NODES ($M=100000$ TUPLES)

Algorithms	$\theta = 0.2$			$\theta = 0.4$			$\theta = 0.6$			$\theta = 0.8$			$\theta = 1$			$\theta = 1.2$		
	30%	20%	10%	30%	20%	10%	30%	20%	10%	30%	20%	10%	30%	20%	10%	30%	20%	10%
GBS	1	1	1	1	2	8	3	4	14	2	3	12	2	2	8	2	2	5
	1	1	6	3	8	22	7	16	71	5	12	64	6	9	49	4	14	21
	3	13	35	5	19	71	6	26	129	14	21	94	13	20	55	9	16	46
pSEQ	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC
	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC
	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC	HC
pCROSS	219	370	HC	143	242	487	50	84	184	9	15	42	2	3	11	1	1	5
	829	1437	HC	597	1045	2449	195	340	876	33	61	183	6	12	55	3	5	23
	1911	HC	HC	1462	2541	HC	478	852	2154	79	136	354	13	24	62	4	12	47
pADA	205	336	HC	146	249	HC	46	88	300	24	58	264	27	82	272	27	70	274
	1564	3691	HC	1173	3056	HC	392	1422	HC	61	332	HC	30	82	HC	34	85	HC
	HC	HC	HC	6578	HC	HC	2656	HC	HC	394	HC	HC	75	HC	HC	40	HC	HC
pBIF	146	198	439	109	145	211	45	66	96	8	14	32	2	2	11	2	2	5
	583	810	1004	402	482	920	170	253	690	38	47	80	7	11	48	7	7	27
	2102	2352	2755	1097	1465	1754	415	635	1431	77	128	258	13	25	57	12	14	44
pSCOUNT	224	359	527	155	255	356	45	72	136	53	73	172	2	3	19	2	2	15
	HC	HC	HC	HC	HC	HC	165	242	667	58	85	205	12	15	51	16	29	81
	HC	HC	HC	HC	HC	HC	367	662	1011	98	156	381	20	27	70	20	34	105

HC: High Cost (when the number of hops $\geq N$; notice that we use sampling without replacement)

It may not be better than the centralized bifocal, but it is comparable. On the other hand, $pBIF$ is not so efficient. The main reason is that it uses function $\text{mult}_T(u)$, which calculates the appearances of value u in the tuples of sampled nodes, instead of function $\text{mult}_R(u)$, which counts the number of tuples in the whole relation R (because cost would be unacceptable, since it would require $O(N)$ hops!).

2) *Precision*: The relative variance for a node-sample size $n=50$, for various values of θ is illustrated in Fig. 6, for $N=1000$ nodes (results are similar for other values of N). GBS has higher precision (i.e. lower relative variance) than the others, except $pSCOUNT$; its relative variance ranges from 0.004 ($\theta=0.2$) to 0.215 ($\theta=0.8$). $pADA$ is not illustrated because its precision is not good (>1.3). $pSEQ$, in turn, does not have a good behavior for $\theta>0.4$ (a relative variance >0.6). $pBIF$ and $pCROSS$ behave almost similarly for $\theta<0.8$, but they both perform very poorly for $\theta<0.6$. $pSCOUNT$ has the highest precision. This means that $pSCOUNT$ is the most predictable of the others, and GBS follows. With respect to precision, there is no evidence in the centralized (self-) join estimation algorithms, thus we cannot compare our results with the centralized versions.

3) *Performance - Messaging Costs*: Performance is measured in terms of the average number of messages/hops sent to select an appropriate number of node-samples, required to reach a specified accuracy.

Table 1 presents the performance costs for all six estimation algorithms and for different Zipfian distributions (different θ s). Performance is expressed in terms of the

number of hops each algorithm needs to achieve high accuracy in SJS estimation, specifically, a %age error less than 30%, 20%, and 10%. Performance is presented with three numbers (three rows per cell) which express the number of hops in different network settings, i.e. for $N=1000, 4000$, and 10000 nodes respectively. This means that, for a total number of tuples equal to 100000 , there will be (on average) 100, 25 and 10 tuples per node.

Table 1 shows that GBS keeps the cost low for all different distributions (i.e. less than 14 hops for a network of 1000 nodes and required accuracy $<10\%$). Only for high θ s ($\theta=1, 1.2$) $pCROSS$ and $pBIF$ have comparable costs with GBS . However, the precision of these two algorithms for high values of θ is lower than GBS (see Fig. 6); in addition, these algorithms are very expensive for the other Zipfian distributions (being typically several-hundred percent worse).

Specifically, a part of Table 1 is presented in Fig. 7, which illustrates the number of hops necessary to achieve an accuracy of %age error < 10 , for all six SJS estimation sampling algorithms and various values of θ , in a network of $N=1000$ and $M=100000$ tuples. It is obvious that GBS requires a significant low number of hops for all different distributions (i.e. in all θ s, less than 14 hops). Only $pCROSS$ and $pBIF$ show similar performance for high θ s ($\theta=1, 1.2$), as already discussed. But for low and medium θ s, $pCROSS$ executes from 32 to more than 1000 hops, and $pBIF$ from 32 to 439 hops; both extremely more expensive than GBS . $pSEQ$ is significantly expensive, and thus, it does not appear in the figure.

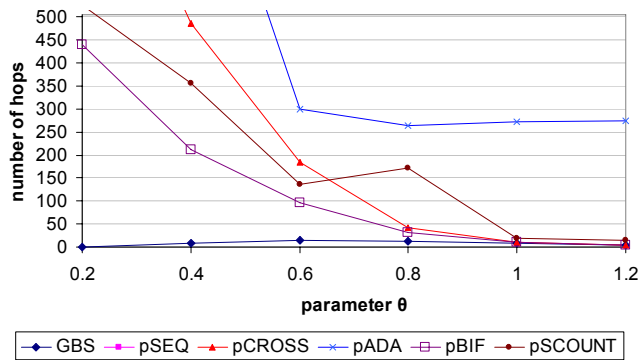


Fig. 7 Performance (expressed by the number of hops) of various SJS estimation sampling algorithms for various θ s, for a %age error < 10, and $N=1000$ ($M=100000$ tuples)

VI. CONCLUSIONS – FUTURE WORK

Estimating SJS in p2p systems is an open problem, which has not been tackled by other researchers. The paper introduced a novel technique to estimate SJS in a p2p data network where attribute values are distributed following Zipf's law, using a statistic metric, the Gini coefficient (G). We further contributed theoretical analyses that relate G with the skew parameters of the Zipfian distributions.

Moreover, the paper develops p2p versions of five well-known SJS estimation techniques. The superiority of the novel G-based algorithm, in terms of accuracy, precision, and messaging performance, is testified by comprehensive experimentation. In fact, the proposed algorithm is the only one to guarantee superior performance in all metrics and for data sets of varying skewness: from nearly-uniform to highly-skewed. In addition, the proposed method ensures estimation accuracy that is similar to that enjoyed by the best techniques in centralized settings, with small performance overheads in a p2p environment.

In addition, the proposed algorithms assume a kind of stratification of tuples into the different nodes. Although it may be easy to implement and less expensive (in terms of hops), it may be not the best approach for achieving high accuracy (i.e. a node does not constitute a random sample). In the future, we plan to investigate other sampling techniques than random walking, which may further improve our estimation technique.

REFERENCES

- [1] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt, "Indexing Data-oriented Overlay Networks," in *Proc. VLDB '05*, 2005, p. 685-696.
- [2] N. Alon, N., P. B. Gibbons, Y. Matias, and M. Szegedy, "Tracking Join and Self-Join Sizes in Limited Storage," in *Proc. PODS*, 1999, p.10-20.
- [3] N. Alon, Y. Matias, and M. Szegedy, "The space Complexity of Approximating the Frequency Moments," in *Proc. STOC*, 1996, p. 20-29.
- [4] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and . Zimmer, "Improving Collection Selection with Overlap Awareness in p2p Search Engines," in *Proc. SIGIR*, 2005, p. 67-74.
- [5] A. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," in *Proc. ACM SIGCOMM*, 2004, p. 353-366.

- [6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *Proc. IEEE INFOCOM*, 1999, vol.1, p. 126-134.
- [7] S. Chaudhuri, R. Motwani, and V. Narasayya, "On Random Sampling over Joins," in *Proc. SIGMOD*, 1999, p. 263-274.
- [8] S. Christodoulakis, "Estimating block transfers and join sizes," in *Proc. SIGMOD*, 1983, p. 40-54.
- [9] C. Damgaard, and J. Weiner, "Describing Inequality in Plant Size or Fecundity," *Ecology*, vol. 81, issue 4, pp. 1139-1142, Apr. 2000.
- [10] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi, "Processing complex aggregate queries over data streams," in *Proc. SIGMOD*, 2002, p. 61-72.
- [11] C. Estan, and J.F. Naughton, "End-biased Samples for Join Cardinality Estimation," in *Proc. ICDE '06*, 2006, p. 20.
- [12] S. Ganguly, P.B. Gibbons, Y. Matias, and A. Silberschatz, "Bifocal Sampling for Skew-Resistant Join Size Estimation," in *Proc. SIGMOD*, 1996, p. 271-281.
- [13] P.B.Gibbons, and Y.Matias, "New Sampling-Based Summary Statistics for Improving Approximate Query Answers," in *Proc. SIGMOD*, 1998, p. 331-342.
- [14] P.J. Haas, J.F. Naughton, S. Seshadri, and L. Stokes, "Sampling-Based Estimation of the Number of Distinct Values of an Attribute," in *Proc. VLDB*, 1995, p. 311-322.
- [15] P.J. Haas, J.F. Naughton, S.Seshadri, and A.N.Swami, "Fixed-Precision Estimation of Join Selectivity," in *Proc. PODS*, 1993, p. 190-201.
- [16] P.J.Haas, J.F.Naughton, and A.N.Swami, "On the Relative Cost of Sampling for Join Selectivity Estimation," in *Proc. PODS*, 1994, p. 14-24.
- [17] P.J.Haas, and A.N. Swami, "Sequential Sampling Procedures for Query Size Estimation," in *Proc. SIGMOD*, 1992, p. 341-350.
- [18] T. Haveliwala, A. Gionis, and P. Indyk, "Scalable Techniques for Clustering the Web," in *Proc. WebDB*, 2000, 129-134.
- [19] W.C. Hou, G. Ozsoyoglu, and E. Dogdu, "Error-constrained COUNT Query Evaluation in Relational Databases," in *Proc. SIGMOD*, 1991, p. 278-287.
- [20] W.C.Hou, G. Ozsoyoglu, and B. Taneja, "Statistical estimators for relational algebra expressions," in *Proc. PODS*, 1988, p. 276-287.
- [21] R.Huebsch, B. Chun, J.M. Hellerstein, B.T. Loo, P. Maniatis, T. Roscoe, S. Sheker, I. Stoica, and A.R. Yumerefendi, "The Architecture of PIER: an Internet-Scale Query Processor," in *Proc. CIDR*, 2005, p. 28-43.
- [22] Y.E. Ioannidis, and V. Poosala, "Balancing Histogram Optimality and Practicality for Query Result Size Estimation," in *Proc. SIGMOD*, 1995, p. 233-244.
- [23] R.Lipton, and J.F. Naughton, "Query Size Estimation by Adaptive Sampling," *Journal of Computer and System Sciences*, vol. 51, issue 1, pp. 18-25, 1995.
- [24] R.J.Lipton, J.F. Naughton, and D.A. Schneider, "Practical Selectivity Estimation through Adaptive Sampling," in *Proc. SIGMOD*, 1990, p. 1-12.
- [25] C. Lynch, "Selectivity estimation and query optimization in large databases with highly skewed distributions of column values," in *Proc. VLDB*, 1988, p. 240-251.
- [26] W.S. Ng, B.C. Ooi, K. Tan, and A. Zhou, "PeerDB: A P2P-based System for Distributed Data Sharing," in *Proc. ICDE*, 2003, p. 633.
- [27] V.Padmanabhan, and L. Qiu, "The Content and Access Dynamic of a Busy Web Site: Findings and Implications," in *Proc. SIGCOMM*, 2000, p. 111-123.
- [28] T.Pitoura, and P.Triantafillou, "Load Distribution Fairness in P2P Data Management Systems," in *Proc. ICDE*, 2007, p. 396-405.
- [29] S.Saroui, P. Gummadi, and S.Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Proc. MMCN*, 2002.
- [30] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proc. SIGCOMM*, 2001, p. 149-160.